

(FILE 'USPAT' ENTERED AT 09:40:41 ON 08 OCT 1998)  
L1 1 S 5668811/PN  
L2 1 S L1 AND SOFTWARE  
L3 1 S L2 AND DRIV?  
L4 15 S ISOCHRONOUS (5A) CHANNEL (P) BUFFER#  
L5 7 S SENDER CLIENT  
L6 0 S LISTENER CLIENT  
L7 4135 S LISTENER  
L8 2 S L4 AND SENDER  
L9 1 S L8 AND CLIENT

=> d 18 1-

1. 5,815,678, Sep. 29, 1998, Method and apparatus for implementing an application programming interface for a communications bus; Gary Alan Hoffman, et al., 395/309, 828 [IMAGE AVAILABLE]

✓ 2. 5,754,789, May 19, 1998, Apparatus and method for controlling point-to-point interconnect communications between nodes; Andreas G. Nowatzky, et al., 395/200.63, 182.1, 200.67, 200.68, 200.78 [IMAGE AVAILABLE]

=> d

1. 5,815,678, Sep. 29, 1998, Method and apparatus for implementing an application programming interface for a communications bus; Gary Alan

DRAWING DESC:

DRWD(8)

FIG. 8 and FIG. 9 contain programming flow diagrams representing **client** and server applications for asynchronous data transport respectively.

DRAWING DESC:

DRWD(9)

FIG. 10 and FIG. 11 illustrate data buffering used by **sender** and receiver application for isochronous data transport respectively.

DRAWING DESC:

DRWD(10)

FIG. 12 and FIG. 13 contain programming flow diagrams representing **sender** and receiver applications for isochronous data transport respectively.

DETDESC:

DETD(10)

The command, **talk**, sends isochronous data from the originator to a specific IEEE 1394 **channel** while **listen** accepts **isochronous** data from a specific **channel**. Any node on the bus can register to listen to this data stream by allocating a user data **buffer** and making a request of the API. Each channel will have only one talker, but possibly multiple listeners. Talkers who wish to stop transmitting data may inform the **isochronous** resource manager to release the **channel** resources to other nodes that may require it.

DETDESC:

DETD(71)

Asynchronous data transfer follows the server-**client** communication model.

DETDESC:

DETD(76)

Asynchronous Communications Model for the **Client** Application

DETDESC:

DETD(80)

FIG. 10 illustrates the data **buffers** for the API **isochronous** transmit command, **talk.sub.-- channel()**:

DETDESC:

DETD(110)

FIG. 11 illustrates the data **buffers** for the API **isochronous** receive command, receive.sub.-- **channel()**:

DETDDESC:

DETD(135)

Isochronous data transfer follows the **sender**--receiver communication model.

DETDDESC:

DETD(136)

Isochronous Communications Model from the **Sender**

DETDDESC:

DETD(137)

FIG. 12 is the flow diagram for the isochronous communication model for the **sender**. After the process starts in step 1200, the number of adapters installed in the system is determined by the get.sub.--. . .

DETDDESC:

DETD(141)

Step . . . from the application via the flush.sub.-- **channel()** call, which will stop any pending isochronous transfers. Step 1226 performs the release.sub.-- **buffer()** calls to free the user level **buffers** allocated by the allocate.sub.-- **buffer()** call in step 1206. Step 1228 releases an **isochronous channel** via the release.sub.-- **channel()** call. Step 1230 frees any internal resources that the application had reserved via the close.sub.-- **adapter()** call. Step 1232 ends. . .

DETDDESC:

DETD(143)

FIG. 13 is the flow diagram for the isochronous communication model for the **sender**. After the program starts in step 1300, the number of adapters installed in the system is determined by the get.sub.--. . .

DETDDESC:

DETD(151)

API Name	Description
allocate.sub.-- <b>buffer()</b>	Allocate a <b>buffer</b> of pinned computer memory
allocate.sub.-- <b>channel()</b>	Allocate an <b>isochronous channel</b>
broadcast.sub.-- 1394()	Broadcast to all devices on local bus segment
close.sub.-- <b>adapter()</b>	Close an adapter and free up memory
flush.sub.-- <b>channel()</b>	Release a <b>buffer</b> used for <b>isochronous</b> reception
get.sub.-- <b>adapter.sub.-- count()</b>	

```

        Get number of interface adapters in computer
get.sub.-- phyid()
        Get PHYSical Layer ID
get.sub.-- WWUID()
        Get WWUID (World-Wide Unique ID)
listen.sub.-- channel()
        Queue a buffer for isochronous reception
lock.sub.-- 1394()
        Asynchronous read or write using a Lock
map.sub.-- 1394.sub.-- space()
        Register address space for a 1394 device
open.sub.-- adapter()
        Open an adapter
read.sub.-- 1394()
        Read asynchronous data from a 1394 device
release.sub.-- buffer()
        Release buffer space
release.sub.-- channel()
        Release an isochronous channel
reset.sub.-- bus()
        Initiate a bus reset
talk.sub.-- channel()
        Queue a buffer for isochronous transmission
unmap.sub.-- 1394.sub.-- space(1)
        Unregister address space for a 1394 device
write.sub.-- 1394()
        Write asynchronous data to a. . .

```

DETDESC:

DETD(222)

---

```

handle  handle of the IEEE 1394 interface previously connected
        by open.sub.-- adapter()
channel isochronous channel
packet.sub.-- count
        number of packets to be received
header  isochronous header
*buffer pointer to data being received

```

---

DETDESC:

DETD(263)

---

```

handle  handle of the IEEE 1394 interface previously connected by
        open.sub.-- adapter()
channel isochronous channel, 0 . . . 63
hdr.sub.-- buf
        buffer containing an array of isochronous header structures,
        one for each packet transmitted
hdr.sub.-- count
        number of header structures in hdr.sub.-- buf
*buffer pointer to data to be transmitted
buffer.sub.-- size
        length of data buffer, in bytes

```

---

--